Scheduling Analysis from Architectural Models of Embedded Multi-Processor Systems

Stéphane Rubini, Christian Fotsing, Frank Singhoff, Hai-Nam Tran, Pierre Dissaux

Scheduling Analysis from Architectural Models of Embedded Multi-Processor Systems

Stéphane Rubini, Christian Fotsing,
Frank Singhoff, Hai Nam Tran
Univ. Bretagne Occidentale, UMR6285,
Lab-STICC, F29200 Brest, France
{rubini,fotsingta,singhoff}@univ-brest.fr

Pierre Dissaux
Ellidiss Technologies
24, quai de la douane
29200 Brest, France
pierre.dissaux@ellidiss.com

ABSTRACT

As embedded systems need more and more computing power, many products require hardware platforms based on multiple processors. In case of real-time constrained systems, the use of scheduling analysis tools is mandatory to validate the design choices, and to better use the processing capacity of the system.

To this end, this paper presents the extension of the scheduling analysis tool *Cheddar* to deal with multi-processor scheduling. In a Model Driven Engineering approach, useful information about the scheduling of the application is extracted from a model expressed with an architectural language called AADL. We also define how the AADL model must be written to express the standard policies for the multi-processor scheduling.

Categories and Subject Descriptors

C.3 [**Special-Purpose and Application-Based Systems**]: Real-time and embedded systems; D.2.4 [**Software Engineering**]: Software/Program Verification—*validation*

General Terms

Performance, Reliability, Verification

Keywords

Real-Time Embedded System, Real-Time Scheduling, Multi-Processor, Model-Driven Engineering, AADL

1. INTRODUCTION

During several decades and until recently, the processing power supplied by sequential processors has grown more and more quickly. The reason of this trend is the improvement in microelectronic technologies, and the internal parallelism of execution through architectural schemes like single or multiple pipelines, out of order instruction issues or speculative executions. The programming model remains the sequential one, but the processors use complex strategies to automatically extract the Instruction Level Parallelism (ILP) available inside the instruction flow.

However, integrated circuit technology and ILP exploitation have reached a state where potential improvements seem to be limited too. Since few years, the answer to that situation is to increase the computing power through the parallel execution of several execution flows. Multi-task applications or systems contain Thread Level Parallelism (TLP) and thus, are naturally adapted to such an execution platform. Parallel multi-processing is today the main way to obtain more powerful computing machines.

In the context of embedded systems, multiple processors may be found even within small embedded systems (for instance in smart phones[18]). Since a long time, complex embedded systems, like avionic or automotive ones, rely on a large set of processors to achieve their computation needs; but they belong rather to the class of the distributed systems, with loosely coupled processing units. The availability of processors or Systems on Chip (SoC) which integrate several tightly coupled processing cores changes this view, and shifts the programming paradigm towards the shared memory multi-processing.

Beyond the speed-up expected from the parallel computing, multi-processor systems are known to have a better energy efficiency (trade-off performances-power). Pollask's rule states that the performances increase is proportional to square root of the increase of complexity [5]. Duplicating a core should give better performance than developing a twice more complex core.

So, considering multi-processing is now a key feature of the hardware platforms intended to implement high performance embedded systems, the software engineering process must deal with parallel computing resources. We focus our study on applications where the software is described by a set of periodic tasks, and a scheduler is in charge of sequencing the task releases on the system's processors according to extra-functional timing requirements (i.e. real-time systems). The scheduling analysis theory [14], and its extension to the multi-processors, are theoretical frameworks that may help the designers to validate their applications.

Another characteristic of embedded systems is that they usually perform specific functions, and so, engineers can optimize them by using dedicated software and hardware.

Hence, as a lot of designs are specific, there is a need for modeling languages which encompass both the software and hardware architecture of the system. Obviously, the models include information about the available processing units.

In this context, the article investigates the ability of the architectural language AADL to model different multi-processor architectures, and to express significant information for controlling a multi-processor scheduling analysis tool, called Cheddar.

The article is structured as follows. The section 2 presents foundation of real-time scheduling analysis on multi-processor systems, and how the Cheddar tool has been adapted to support such kind of analysis. The section 3 defines the architecture of the parallel execution platforms that we consider for the scheduling analysis. The section 4 binds this analysis to a model driven development process. In the last part, after the presentation of related works, we conclude and give some perspectives.

2. MULTI-PROCESSOR SCHEDULING ANALYSIS WITH THE CHEDDAR TOOL

We consider real-time applications which are modeled by a set of periodic tasks. For schedulability analysis, each task T_i is usually defined by four temporal parameters [14]: its first release time r_i, its Worst Case Execution Time (WCET) C_i which is the highest computation time of the task, its relative deadline D_i which is the maximum acceptable delay between the release and the completion of any instance of the task, and its period P_i which is the fixed delay between two successive task release time. A task consists of an infinite set of instances (or jobs) released at times $r_i + k * P_i$, where $k \in \mathbb{N}$.

From this task model, the community has proposed various ways to assess the schedulability of a real-time application. Each schedulability analysis assumes a scheduling algorithm which decides at each time the jobs that have to be run on the processors. In the sequel, we first give an outline of real-time multiprocessor scheduling algorithms. Then, we show how we have implemented them in the Cheddar tool.

2.1 Multi-processor scheduling

Multi-processor scheduling is used to support multi-programming (large number of independent jobs), or to support parallel programming (dependent jobs with multiple exchanges). Given a set of jobs and a set of processors one of the question a multi-processor scheduling algorithm has to answer is how jobs can be assigned to processors [8].

We distinguish two classes of multi-processor scheduling:

- Global scheduling [3][9]. In this approach, any instance of any task may be executed on any processor. A job may halt its execution on one processor and resume it on a different processor.

  Consequently, this approach assumes an execution environments which allow task migrations. Classically, a task can migrate at any time, or migration can be limited to job activation (e.g. when a task job is started on a given processor, it is supposed to complete on the same processor).

  This class of scheduling algorithms suffers from high run-time overhead as the number of pre-emptions and migrations can increase significantly.

- Partitioned scheduling. With partitioned scheduling, each task can only be executed on a dedicated processor and task migrations are not allowed.

  A significant advantage of partitioned scheduling is that it is well-understood since schedulability analysis can be verified with uniprocessor schedulability analysis methods. Then, classical scheduling such as EDF, RM [14] can be reused.

  But, finding an optimal assignment of tasks to processors is equivalent to a bin-packing problem, which is known to be NP-hard [13]. However, many polynomial-time heuristics have been proposed for solving this problem [10] (First Fit, Best Fit, ...).

Each of these approaches described above have been extended in various studies.

2.2 Cheddar and multi-processor scheduling support

Cheddar is a schedulability tool that implements classical schedulability methods for uniprocessor systems [17]. To support scheduling analysis of multi-processor architectures, we have extended Cheddar with both partitioned and global scheduling features.

As a partitioned system can be analyzed as a set of uniprocessor subsystems, most of the Cheddar features can be used to assess schedulability of these system. Cheddar uniprocessor analysis features only have been extended with some simple bin-packing algorithms in order to define processor's tasks. The currently supported bin-packing algorithms are RM Next Fit, First Fit, Best Fit, Small tasks and General Tasks [7].

For scheduling analysis with global scheduling, Cheddar offers less features. For this class of multiprocessor architectures, Cheddar only offers scheduling analysis by scheduling simulation. The current Cheddar implementation proposes a global scheduling algorithm for each uniprocessor algorithm (e.g. global DM/EDF/RM/LLF/...).

3. MULTI-PROCESSING IMPLEMENTATION

In this section, we describe different implementations of the concept of multi-processing. These implementations differ by the resources in the architecture which are shared when executing the instruction flows. The figure 1 illustrates the major, although not mutually exclusive, solutions: multi-processors, multi-cores and multi-threading.

Historically, multi-processor architectures have been the first solution. The processors are located on different integrated circuits, and share the memory. Memory and bus contention is a major concern in this kind of architecture, and the off-chip implementation of the buses limits the field of implementations that could be used to deal with this problem.

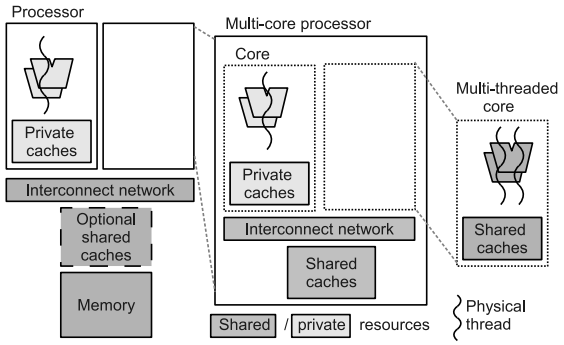

Figure 1: Architectural localization of the multi-processing capabilities. The ability of executing multiple threads may be provided at different levels: by the coupling of processor chips, inside a "processor" chip or inside the data-path.

The off-chip buses are also known to consume a significant part of overall power budget [4].

Today's multi-core processors are only integrated implementations of multi-processor systems. But, the technical constraints are not the same, and more efficient solutions may be developed to interconnect the cores and the memory hierarchy inside a chip. Multiplexed bus, crossbar switches or Network-On-Chip are examples of interconnect architectures that allow parallel transactions between cores, memories, or I/O devices.

Multi-threading strategies have been introduced to increase the usage efficiency of processing units inside the pipeline of the super-scalar processors. When the ILP is too limited and/or the execution flows are stalled, waiting for the end of memory requests, processing units are available for executing instructions issued from another threads.

However, the concept of multi-threading inherently leads to the sharing of the processing units of a core/processor. In the general case, the starting up of an additional thread impacts the performances of the other threads running on the same processor. This implicit interference between threads[1] is very difficult to quantify, because it depends on the set of instructions, coming from the candidate threads, that can be issued at a given time towards the processor pipelines.

The next section describes how the model of a system can take such architectures into consideration, and its exploitation to drive a multi-processor scheduling analysis.

4. SCHEDULING ANALYSIS FROM AADL MODELS

The figure 2 shows the tool chain which drives the scheduling analysis.

OSATE, *Stood* or *ADELE* are textual and graphical model editors for AADL. *AADLInspector*[2] is a model processing framework that is used to analyze textual AADL specifications. It includes a set of static rule checkers and bridges for remote verification tools, like *Cheddar*, or *Marzhin*[3]. Finally, *Cheddar*[4] is a real-time scheduling tool.

Figure 2: Analysis tool chain

In this section, after a short presentation of the AADL language, we discuss the deployment schemes for different multi-processor scheduling strategies. Next, the AADL model of a system, and a scheduling analysis is shown as an example.

4.1 AADL

The Architecture Analysis & Design Language (AADL) is a SAE standard (AS-5506), first published in 2004 [12]. AADL is a modeling language for description and analysis of system architecture in terms of components and their interactions. It allows the modeling of software and execution platform components. The deployment of a software application on an execution platform is specified through *binding* properties.

An AADL component is defined by a declaration and implementations. Each component relies on a category. Categories of components are related to software entities, like *process*, *thread* or *data*, and hardware entities like *processor*, *memory*, *bus* or *device*. Each component may have several attributes called *properties*. The AADL standard includes a large set of pre-declared properties to model system characteristics. Moreover, new properties can be defined to precisely describe the expected system.

4.2 Modeling and Assignment of computing units

As noticed before, AADL represents the deployment of an application by binding properties, and hence associates software entities to execution resources. For instance, the following property assigns the execution of a thread *ta*, belonging to the process *as*1, to a processor *p*1.

```
Actual_Processor_Binding=>(reference(p1))
                          applies to as1.ta;
```

Now, we will explain how this binding mechanism may be used in the multi-processing context.

Partitioned scheduling. To deal with partitioned scheduling, a list of software threads may be assigned to their targeted processors by duplicating the processor bindings for all of them. As AADL defines a processor as an executing

[1]The same problem exists when threads share other resources like memory buses for instance.

[2]*AADLInspector* and *Stood* are products of Ellidiss Technologies (`www.ellidiss.com`)

[3]*Marzhin*, a multi-agent based simulator, is a product of Virtualys and Ellidiss Technologies.

[4]*Cheddar* may be downloaded from `http://beru.univ-brest.fr/~singhoff`

platform including a scheduler that manages the sharing of its computing resources, the system is modeled as a set of nodes with their associated local scheduling policy . Nodes can be connected by a bus component.

We expect that one AADL processor should be associated to one processing unit, i.e. hardware executing one instruction flow, regardless whether this processing unit constitutes a core of a processor, or a processor as a whole (we postpone the case of multi-threading later in this section); hence, an AADL processor includes the private resources dedicated to an instruction flow, mainly the execution pipelines and the private cache memories (cf figure 1). Such a modeling guideline aims to exclude of the "processor" all the shared resources between them, in order to define independent subsystems when they do not need to interact through external resources. Shared resources are often a bottleneck in multi-processors or multi-cores and the models should highlight them.

Global scheduling. A global scheduler maintains a system wide queue of ready tasks and considers a set of processors to execute them. An AADL processor deals with a single hardware execution flow at a given time, and cannot be used to model a set of processors. A first approach is to group the target processors into a system component, and to bind the threads to this multi-processing system (*smp* in the example below).

```
Actual_Processor_Binding=>(reference(smp))
    applies to as1.ta;
```

An alternative binding method enumerates explicitly the set of processors involved in the scheduling.

```
Actual_Processor_Binding=>
  (reference(smp.p1),reference(smp.p2))
    applies to as1.ta;
```

An AADL processor includes a scheduler, of which the type is specified by the standard property `Scheduling_Protocol`. In the case of global scheduling, the value of this property must be the same for all the processors. To remove this consistency rule, the AADL standard could be adapted to associate a global scheduling method to all the processing units available within a system component.

Multi-threading. Multi-threaded processors or cores allow the parallel execution of few tasks. The AADL hardware model should represent each physical thread by a processor component.

A pessimistic performance scaling factor may be defined, according to the maximal number of physical threads per core, in order to take into account the performance provided by a physical thread in relation with the mono-thread core performance. The AADL standard property `Scaling_Factor` and `Reference_Processor` allow to specify the scaling factor with respect to a reference processor, and can be used in that goal.

Now, we will illustrate this guideline by modeling a simple image processing application on a multi-core execution platform.

4.3 Example

We have chosen the LEON4_DEMO_ASIC of *AEROFLEX Gaisler* [1] as a target platform for our case study. The key features of this Multi-Processor SoC (MPSoC) are:

- Dual LEON4 cores running at 200 MHz with symmetric multi-processing support. An instruction cache and a data cache are associated to each core. The LEON4 core does not provide multi-threading capabilities.
- Many on-chip devices and IO connections: USB-2.0 host/device, SVGA, Ethernet, PCI, I2C, CAN, ...

Figure 3 shows the general architecture of the SoC, structured around three types of buses: (1) the processor memory bus, a 64-bit multiplexed synchronous AMBA AHB at 200 MHz, (2) the 32-bit AHB Fast I/O bus at 100 MHz, and (3) two 32-bit low latency asynchronous APB I/O buses.

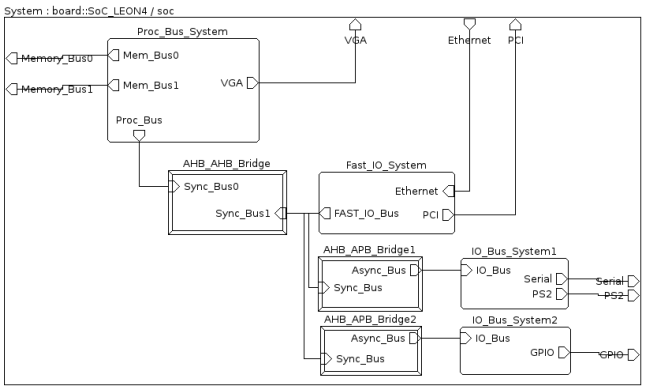

Figure 3: Simplified architecture of the SoC, modeled with the AADL graphical notation. Rounded rectangles and "double" rectangles represent respectively subsystems and devices.

Bus utilization is a major concern in the design of a SoC; bus bandwidth and device communication requirements must be balanced. So, the organization of the AADL model follows the bus architecture of the SoC.

The next listing describes, with the AADL textual syntax, the simplified model of the subsystem connected by the processor memory bus. The component implementation `Core.Leon4` is not detailed here; mainly it defines some processor properties, and characteristics of its private cache level.

```
system Proc_Bus_System
features
  Proc_Bus : provides bus access;
  Mem_Bus0 : provides bus access; -- bank 0
  Mem_Bus1 : provides bus access; -- bank 1
  VGA      : provides bus access;
end proc_bus_system;
system implementation Proc_Bus_System.SoC_Leon4
subcomponents
```

```
   Core1     : processor Core.Leon4 {
    Scheduling_Protocol=>Highest_Priority_First;
   }
   Core2     : processor Core.Leon4 {
    Scheduling_Protocol=>Highest_Priority_First;
   };
   FSB       : bus Communication_Bus.AHB;
   DDR2_Ctrl: device Mem_Ctrl.DDR2;
   Fb        : device Framebuffer.VGA;
connections
   bus access FSB <-> Core1.Proc_Bus;
   bus access FSB <-> Core2.Proc_Bus;
   bus access FSB <-> Proc_Bus;
   ...
end Proc_Bus_System.SoC_Leon4;
```

In association with the hardware model, AADL allows us to specify the software architecture. In our example, we consider a simple image processing process; figure 4 outlines the processing steps, implemented by three periodic tasks.

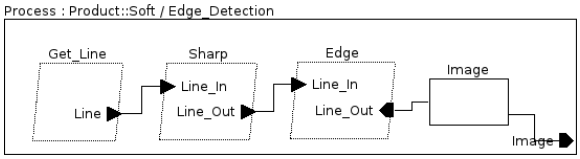

Figure 4: Software architecture. Rectangles and dashed parallelograms represent respectively shared data and execution threads. Threads communicate through data ports, which define data dependencies.

The AADL standard supplies a set of properties to qualify the behavior of real-time tasks. These properties give the information to define the parameters of the task model presented in the beginning of section 2.

```
process Edge_Detection     end Edge_Detection;
process implementation Edge_Dectection.impl
subcomponents
   Get_Line : thread IO {
     Dispatch_Protocol => Periodic;
     Period => 20 us;
     Compute_Execution_Time => 2 us .. 4 us;
     Deadline => 20 us;
     Priority => 10;
   };
   Sharp : ...
end Edge_Dectection.impl;
```

Finally, the root of the model hierarchy brings together the hardware and the software view of the embedded system in a system component.

```
system Product;   end Product;
system implementation Product.impl
subcomponents
   Hard : system Soc.Soc_Leon4;
   Soft : process Edge_Detection.impl;
   ...
properties
   -- deployment (see below)
end Product.impl;
```

The deployment properties complete the model and bind the software entities to components of the execution platform.

The binding properties below define that the three threads of our application may be executed on the two cores of the SoC. We implement a global scheduling multi-processor strategy here.

```
Actual_Processor_Binding =>
   (reference(Hard.Proc_Bus_System))
         applies to Soft.Get_Line;
Actual_Processor_Binding =>
   (reference(Hard.Proc_Bus_System))
         applies to Soft.Sharp;
-- or with an explicit list of processors
Actual_Processor_Binding =>
   (reference(Hard.Proc_Bus_System.Core1),
    reference(Hard.Proc_Bus_System.Core2))
         applies to Soft.Edge;
```

The Gantt diagram shown in figure 5 is the partial result of a simulation performed by the Cheddar tool. Information provided by the AADL model (component hierarchy, property values, and the deployment bindings) are transformed into Cheddar's Architecture Description Language (ADL). The three time-lines represent the instants when tasks are released.

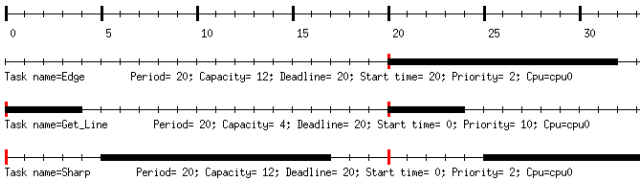

Figure 5: Result of a Cheddar simulation. The Cheddar ADL groups in a "Cpu" the set of processing units managed by a global scheduling algorithm.

5. RELATED WORKS

Other works focus on the integration of a scheduling analysis in a MDE approach. They differ from our proposition by the expressiveness of the ADL and the analysis method that has been chosen.

UML/Modeling and Analysis of Real-Time and Embedded System (MARTE) profile currently supports mono and multi-processor scheduling algorithms, but only for a partitioned approach. [15] has proposed various updates for MARTE metamodels of specialization and generalization stereotype in order to support global scheduling approaches, allowing task migrations. Those changes allow a schedulable resource to be executed on different computing resources in the same period.

The work in [2] describes an approach to combine MARTE and EAST-ADL2 to overcome EAST-ADL2 limitation of notions for modeling the timing features. EAST-ADL2 is an architecture description language defined as a domain specific language for the development of automotive electronic systems. The MAST toolset is integrated in the MDE process to perform the scheduling analysis.

An approach to extend the AADL standard properties to support the modeling and specification of embedded multi-core system was also proposed in [11]. Analysis is performed

with a Monte-Carlo and scheduling simulation mixed approach.

In [6], a verification framework for schedulability of multicore systems, called MoVES, is described. MoVES provides a simple specification language to define a system. It can model software components and execution platform; however, the architecture seems to be restricted to only one processor bus. The analysis method is based on timed automata.

6. CONCLUSION

This article has shown how scheduling analysis may be handled in a MDE process. The AADL language provides the means to express the basic information required to control a multi-processor scheduling tool. The availability of multiples processing units extend the design space and engineers need help at the early stages of the design to check their choice about their assignment to the tasks.

For our future work, we plan an extension of the Cheddar simulation framework to consider uniform processors, i.e. processors with equal capabilities but different speeds. Especially, this feature will help to integrate some of the effects related to the multi-threaded processors in the results. After that, a key point in the multi-processor analysis is to consider the sharing of resources, like buses or shared caches, which introduces implicit inter-processor dependencies. Extensions of the Cheddar tool to deal with some of these aspects will be useful, and once again precise architectural models of the computing and memory resources [16] will be required.

7. ACKNOWLEDGMENTS

This work is done in the context of the SMART project, in collaboration with *Ellidiss Technologies* and *Virtualys*. SMART is supported by the *Conseil Régional de Bretagne* and OSEO. Cheddar is supported by the *Conseil Régional de Bretagne* and Ellidiss Technologies.

8. REFERENCES

[1] Aeroflex Gaisler. *Dual core LEON4 SPARC V8 Processor LEON4-ASIC-DEMO, Data Sheet and User's Manual*, May 2011.

[2] S. Anssi, S. Tucci-Pergiovanni, C. Mraidha, A. Albinet, F. Terrier, and S. Gérard. Completing EAST-ADL2 with MARTE for enabling scheduling analysis for automotive applications. *Embedded Real Time Software and Systems, Toulouse, France*, 2010.

[3] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel. Proportionate progress: A notion of fairness in resource allocation. *Journal of Algorithmica*, 15(6):600–625, 1996.

[4] K. Basu, A. Choudhary, J. Pisharath, and M. Kandemir. Power protocol: reducing power dissipation on off-chip data buses. In *Proceedings of the 35th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 345–355, June 2002.

[5] S. Borkar. Thousand core chips: a technology perspective. In *Proceedings of the 44th annual Design Automation Conference*, pages 746–749. ACM, 2007.

[6] A. Brekling, M. R. Hansen, and J. Madsen. MoVES-a framework for modelling and verifying embedded systems. In *Proceedings of the IEEE International Conference on Microelectronics*, pages 149–152, 2009.

[7] A. Burchard, J. Liebeherr, Y. Oh, and S. Son. Assigning real-time tasks to homogeneous multiprocessor systems. *IEEE Transactions on Computers*, 44(12):1429–1442, December 1995.

[8] J. Carpenter, S. Funk, P. Holman, A. Srinivasan, J. Anderson, and S. Baruah. A categorization of real-time multiprocessor scheduling problems and algorithms. *Leung, J. Y.-T., editor, Handbook of Scheduling: Algorithms, Models, and Performance Analysis. CRC Press LLC.*, 2003.

[9] H. Cho, B. Ravindran, and D. Jensen. An optimal real-time scheduling algorithm for multiprocessors. In *Proceedings of the 27th IEEE International Real-Time Systems Symposium*, pages 101–110. IEEE, 2006.

[10] E. G. Coffman, G. Galambos, S. Martello, and D. Vigo. Bin packing approximation algorithms: Combinational analysis. *Kluwer Academic Publishers*, 1998.

[11] M. Deubzer, M. Hobelsberger, J. Mottok, F. Schiller, R. Dumke, M. Siegle, U. Margull, M. Niemetz, and G. Wirrer. Modeling and simulation of embedded real-time multicore systems. In *Proceedings of the 3rd Embedded Software Engineering Congress*, pages 228–241, 2010.

[12] P. Feiler, B. Lewis, and S. Vestal. The SAE AADL standard: A basis for model-based architecture-driven embedded systems engineering. In *Workshop on Model-Driven Embedded Systems*, May 2003.

[13] J. Y.-T. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance evaluation*, 2(4):237–250, 1982.

[14] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environnment. *Journal of the Association for Computing Machinery*, 20(1):46–61, Jan. 1973.

[15] A. Magdich, Y. H. Kacem, A. Mahfoudhi, and M. Abid. A MARTE extension for global scheduling analysis of multiprocessor systems. In *Proceedings of the 23rd IEEE International Symposium on Software Reliability Engineering*, pages 371–379, 2012.

[16] S. Rubini, F. Singhoff, and J. Hugues. Modeling and verification of memory architectures with AADL and REAL. In *Sixth IEEE International Workshop on UML and AADL*, pages 338–343, USA, April 2011.

[17] F. Singhoff, J. Legrand, L. Nana, and L. Marcé. Cheddar: a Flexible Real-Time Scheduling Framework. *ACM SIGAda Ada Letters, ACM Press, New York, USA*, 24(4):1–8, Dec. 2004.

[18] C. Van Berkel. Multi-core for mobile phones. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 1260–1265. European Design and Automation Association, 2009.